Mechatronics 23 (2013) 541–547

Contents lists available at SciVerse ScienceDirect

Mechatronics

journal homepage: www.elsevier.com/locate/mechatronics

Technical note

A generalised solution for generating stepper motor speed profiles in real time

P.J. Siripala, Y. Ahmet Sekercioglu *

Department of Electrical and Computer Systems Engineering, Monash University, Melbourne, Australia

ARTICLE INFO

Article history:
Received 25 October 2012
Accepted 14 April 2013
Available online 2 June 2013

Keywords:
Stepper-motor control
Real-time control
Mobile robotics
Acceleration

ABSTRACT

We present a new low-complexity algorithm for controlling a stepper motor to rotate with constant acceleration or deceleration. It can be operated on a basic low-end microcontroller and does not require any data tables. Also, it does not have the limitation of the motor having to start from standstill. The algorithm was tested using a small robot programmed to move and change acceleration 'on the run'. The theoretical foundations for further improving the timing accuracy of the algorithm were also derived.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

Stepper Motors are used in many small devices which require relatively fast speeds or high torques. The physical architecture of the motor also allows for the motor's position to be controlled accurately without the need for close-loop feedback techniques. For some purposes, the use of wheel encoders can then be neglected – particularly for static loads [2].

There is currently ample research on operating motors with constant speed. The practical applications for constant speed are obvious. Most techniques – if not all – use close-loop feedback techniques. Mukherjee et al. in [3] provides a solution for self-controlled synchronous motors. Likewise Hu et al. in [4] provides a solution for brushless DC motors. Both techniques however provide no method to traverse between two different constant speeds (i.e. accelerate/decelerate) in a smooth and consistent manner.

There are also many applications which require constant acceleration or deceleration in its own right. Athani in [5] mentions their heavy industrial use in Computer Numerical Control Systems. Other uses include high speed pick and place equipment. In the field of lasers and optics, they are frequently used in precision positioning equipment too. Prior to Austin's approach in [1], the timing of the step pulses were considered too complicated to calculate in real time [1]. The solution was either to use precompiled timing data for the desired acceleration rates or more powerful microcontrollers. The first solution is inflexible and the second solution conflicts with the general aim to minimise use of unnecessary resources.

Austin's approach however assumes that acceleration begins when the stepper motor is at standstill. This is an unacceptable limitation to many projects. This paper presents a generalisation of Austin's approach so that a stepper motor can accelerate or decelerate at any desired constant rate irrespective of the starting angular velocity.

2. Definitions and preliminary remarks

To understand why it had been difficult to create constant acceleration on stepper motors accurately before [1], the nature of the stepper motor must be understood.

When you accelerate a non-stepper motor, at regular time intervals you make it rotate a given amount. If the desired acceleration rate is 1 rad/s^2, within the duration of the 1st second the motor would be programmed to rotate 1 radian. Within the 2nd second, the motor would be programmed to rotate 2 radians and so on and so forth.

A stepper motor moves in fixed angle intervals unlike typical motors. Each 'step' rotates the motor by an angle of α radians. For a typical hobby stepper motor, α is 1.8 degrees. This means 200 steps must fire in order to rotate a full 360 degrees.

Consequently, this discrete nature means a stepper motor cannot rotate 2 degrees. If a rotation of 2 degrees is required, you must either rotate 1.8 degrees or rotate $2 \times 1.8 = 3.6$ degrees.

With a stepper motor, due to its discrete rotatable angles, it is not viable to rotate the motor the required angle to regular time intervals. Rather, Austin's approach varies the time intervals to regular α intervals. This paper utilises the same principle.

Many basic microcontrollers come with a timer module which increments a 16-bit counter value at a fixed frequency f_t. Typically

* Corresponding author. Tel.: +61 3 9905 3503; fax: +61 3 9905 3454.
E-mail address: ASekerci@ieee.org (Y.A. Sekercioglu).

0957-4158/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.mechatronics.2013.04.004

the fixed frequency is determined by the external clock of the microcontroller. When the counter value reaches an arbitrarily set comparison value c, it triggers an interrupt and then resets back to zero. Since the counter value is reset back to zero, the process repeats and the interrupt is fired at a regular interval. The programmer can execute code in the interrupt (i.e. send step pulse to the motor to rotate it by α degrees) knowing that it will execute more or less at a regular interval.

The time intervals between each step pulse are varied by changing the comparison value c of the timer module.

$$\delta t = \frac{c}{f_t} \tag{1}$$

Since f_t is fixed, the comparison value c is changed to get the desired δt time interval to rotate the stepper motor α radians.

It can also be seen that the motor angular speed ω at any given time is:

$$\omega = \frac{\alpha f_t}{c} \tag{2}$$

This paper derives a new time-interval-update function that updates c allowing for acceleration variations irrespective of the initial angular velocity ω_i (and thus allowing for changes in the acceleration rate in real-time).

3. Theoretical derivation

From Fig. 1, it can be seen that $\omega(\tau) = \bar{\dot{\omega}}\tau + \omega_i$. To work out the total angle θ rotated from $\tau = 0$ to an arbitrary time t,$\omega(\tau)$ must be integrated. The total angle rotated must equal a multiple of α for a stepper motor.

$$\int_0^t \omega(\tau)d\tau = \left[\frac{1}{2}\bar{\dot{\omega}}\tau^2 + \omega_i\tau\right]_0^t = \frac{1}{2}\bar{\dot{\omega}}t^2 + \omega_i t = n\alpha, \quad \text{where } n \in \mathbb{Z}^+ \tag{3}$$

This necessarily implies that t cannot be any arbitrary $\mathbb{R}$ but must be at discrete intervals denoted t_n for each corresponding n.

$$\frac{1}{2}\bar{\dot{\omega}}t_n^2 + \omega_i t_n - n\alpha = 0$$
$$t_n = \frac{-\omega_i \pm \sqrt{\omega_i^2 - 4(\frac{1}{2}\bar{\dot{\omega}})(-n\alpha)}}{2(\frac{1}{2}\bar{\dot{\omega}})} = \frac{-\omega_i + \sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}}{\bar{\dot{\omega}}} \quad \text{since } t_n \geqslant 0 \tag{4}$$

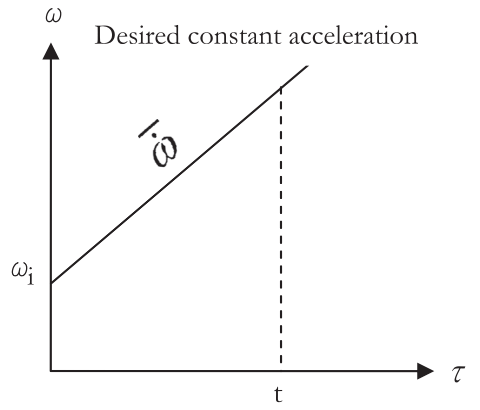

Fig. 1. Angular velocity profile.

The exact timer comparison value c_n required to program the required delay between the nth and $(n + 1)$th pulse $(n \geqslant 0)$ is according to Eq. (1):

$$\begin{aligned} c_n &= f_t(t_{n+1} - t_n) \\ &= f_t\left\{\frac{\left[-\omega_i + \sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n+1)\alpha}\right] - \left[-\omega_i + \sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}\right]}{\bar{\dot{\omega}}}\right\} \\ &= \frac{f_t}{\bar{\dot{\omega}}}\left\{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n+1)\alpha} - \sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}\right\} \end{aligned} \tag{5}$$

It should also be noted that when $\omega_i = 0$ (acceleration from standstill), Eq. (5) simplifies to Eq. (8) in [1]:

$$c_n = f_t\sqrt{\frac{2\alpha}{\bar{\dot{\omega}}}}(\sqrt{n+1} - \sqrt{n}) = c_o(\sqrt{n+1} - \sqrt{n}) \tag{6}$$

The c_0 value (i.e. when $n = 0$) is important further along the derivation. It should be noted now however:

$$c_0 = \frac{f_t}{\bar{\dot{\omega}}}\left\{-\omega_i + \sqrt{\omega_i^2 + 2\bar{\dot{\omega}}\alpha}\right\} \tag{7}$$

Eq. (5) with two square roots is computationally expensive for basic low-end microcontrollers to calculate in real-time. It can be approximated using Second-Order Newton Expansion:

$$(1 + x)^{\Psi} = 1 + \Psi x + \frac{\Psi(\Psi - 1)}{2!}x^2 + \cdots \tag{8}$$

3.1. Approximation

$$\begin{aligned} \frac{c_n}{c_{n-1}} &= \frac{\left\{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n+1)\alpha} - \sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}\right\}}{\left\{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n)\alpha} - \sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n-1)\alpha}\right\}} \\ &= \frac{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}\left[\frac{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n+1)\alpha}}{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}} - 1\right]}{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}\left[1 - \frac{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n-1)\alpha}}{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}}\right]} \\ &= \frac{\frac{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n+1)\alpha}}{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}} - 1}{1 - \frac{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}(n-1)\alpha}}{\sqrt{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}}} \\ &= \frac{\sqrt{\frac{\omega_i^2 + 2\bar{\dot{\omega}}(n+1)\alpha}{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}} - 1}{1 - \sqrt{\frac{\omega_i^2 + 2\bar{\dot{\omega}}(n-1)\alpha}{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}}} \\ &= \frac{\sqrt{1 + \frac{2\bar{\dot{\omega}}\alpha}{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}} - 1}{1 - \sqrt{1 - \frac{2\bar{\dot{\omega}}\alpha}{\omega_i^2 + 2\bar{\dot{\omega}}n\alpha}}} \end{aligned} \tag{9}$$

Now applying Second-Order Newton Expansion to Eq. (9) using the identity in Eq. (8) leads to:

$$\frac{c_n}{c_{n-1}}=\frac{\left[1+\frac{1}{2}\left(\frac{2\bar{\omega}\alpha}{\omega_i^2+2\bar{\omega}n\alpha}\right)+\frac{\left(\frac{1}{2}\right)\left(\frac{-1}{2}\right)}{2}\left(\frac{2\bar{\omega}\alpha}{\omega_i^2+2\bar{\omega}n\alpha}\right)^2\right]-1}{1-\left[1+\frac{1}{2}\left(\frac{-2\bar{\omega}\alpha}{\omega_i^2+2\bar{\omega}n\alpha}\right)+\frac{\left(\frac{1}{2}\right)\left(\frac{-1}{2}\right)}{2}\left(\frac{-2\bar{\omega}\alpha}{\omega_i^2+2\bar{\omega}n\alpha}\right)^2\right]}$$
$$=\frac{\frac{\bar{\omega}\alpha}{\omega_i^2+2\bar{\omega}n\alpha}-\frac{1}{2}\left(\frac{\bar{\omega}^2\alpha^2}{(\omega_i^2+2\bar{\omega}n\alpha)^2}\right)}{\frac{\bar{\omega}\alpha}{\omega_i^2+2\bar{\omega}n\alpha}+\frac{1}{2}\left(\frac{\bar{\omega}^2\alpha^2}{(\omega_i^2+2\bar{\omega}n\alpha)^2}\right)} \tag{10}$$
$$=\frac{2\omega_i^2+4\bar{\omega}n\alpha-\bar{\omega}\alpha}{2\omega_i^2+4\bar{\omega}n\alpha+\bar{\omega}\alpha}$$
$$=\frac{2\omega_i^2+4\bar{\omega}n\alpha+\bar{\omega}\alpha-\bar{\omega}\alpha-\bar{\omega}\alpha}{2\omega_i^2+4\bar{\omega}n\alpha+\bar{\omega}\alpha}$$
$$=1-\frac{2\alpha\bar{\omega}}{2\omega_i^2+4\bar{\omega}n\alpha+\bar{\omega}\alpha}$$

After rearranging to obtain the new comparison value as a function of the old comparison value:

$$c_n=c_{n-1}-\frac{2\alpha\bar{\omega}c_{n-1}}{2\omega_i^2+\alpha\bar{\omega}(4n+1)} \tag{11}$$

This is an important result. It is the new time-interval-update function which does not require the starting angular velocity to be zero. It is also recursive in nature which is less computationally expensive than otherwise.

It should also be noted that when $\omega_i = 0$ (acceleration from standstill), Eq. (11) simplifies to Eq. (12) in [1]:

$$c_n=c_{n-1}-\frac{2c_{n-1}}{4n+1} \tag{12}$$

Eq. (11), whilst simple and easy to implement, is not necessarily accurate since it relies on a Second-Order Newton Expansion approximation.

Austin in [1] tabulates the accuracy of his approach for $\frac{c_n}{c_{n-1}}$, remembering that his results are based on the assumption that $\omega_i = 0$:

Austin defines Relative Error as relative error $(n)=\frac{\text{Approx }(n)-\text{Exact }(n)}{\text{Exact }(n)}$ (Table 1).

Austin notes that the relative error is unacceptably high for $n = 1$. An unacceptably high relative error would obviously be a greater concern in a generalisation of Austin's approach.

To rectify the problem, it is suggested that when $\omega_i = 0$, Eq. (7) (also Eq. (7) in [1]) be multiplied by a magic multiple 0.676. Austin gives no explanation or clues to where it comes from:

$$c_0=0.676f_t\sqrt{\frac{2\alpha}{\bar{\omega}}} \tag{13}$$

Table 1
Relative error in Austin's time-interval-update function [1].

Step n	Exact (9)	Approx (11)	Relative error
1	0.4142	0.6000	0.4485
2	0.7673	0.7778	0.0136
3	0.8430	0.8462	0.00370
4	0.8810	0.8824	0.00152
5	0.9041	0.9048	7.66E−04
6	0.9196	0.9200	4.41E−04
10	0.9511	0.9512	9.42E−05
100	0.9950	0.9950	9.38E−08
1000	0.9995	0.9995	9.37E−11

A more complex version of the magic multiple is needed for a generalisation of Austin's approach.

4. Finding the magic multiple

It is not necessarily reasonable to assume that Eq. (11) is a *good* approximation for Eq. (5). Since Eq. (11) is a $f(\alpha,\bar{\omega},\omega_i)$, there are many degrees of freedom for inaccuracies so applying a pre-determined constant magic multiple is not generally feasible.

Before exploring the nature of the magic multiple, firstly the exact and approximate functions need to be clearly defined.

The ***exact*** **time-interval-update** function is from Eq. (5) and Eq. (7):

$$c_n=exact(n)=\begin{cases}\frac{f_t}{\bar{\omega}}[-\omega_i+\sqrt{\omega_i^2+2\bar{\omega}\alpha}], & n=0\\ \frac{f_t}{\bar{\omega}}[\sqrt{\omega_i^2+2\bar{\omega}(n+1)\alpha}-\sqrt{\omega_i^2+2\bar{\omega}n\alpha}], & n>0\end{cases} \tag{14}$$

The ***approximate*** **time-interval-update** function is from (11). Although the algorithm relies on the recursive function, the closed function is also noted.

$$c_n=approx(n)$$
$$=\begin{cases}exact(0)=\frac{f_t}{\bar{\omega}}\left[-\omega_i+\sqrt{\omega_i^2+2\bar{\omega}\alpha}\right], & n=0\\ approx(n-1)-\frac{2\alpha\bar{\omega}\times approx(n-1)}{2\omega_i^2+\alpha\bar{\omega}(4n+1)}, & n>0\end{cases}$$
$$=\begin{cases}exact(0)=\frac{f_t}{\bar{\omega}}\left[-\omega_i+\sqrt{\omega_i^2+2\bar{\omega}\alpha}\right], & n=0\\ exact(0)\times\prod_{k=1}^{n}\frac{2\omega_i^2+\alpha\bar{\omega}(4k-1)}{2\omega_i^2+\alpha\bar{\omega}(4k+1)}, & n>0\end{cases} \tag{15}$$

It is assumed that Austin hypothesised that the relative error for all $n \geqslant 0$ is defined irrespective of ω_i and $\bar{\omega}$ (provided $\bar{\omega}\neq 0$). It is also assumed that Austin hypothesised that for all $n > 0$ the $\lim_{n\to\infty}$ relative error (n) = some constant $\neq 0$. It is further assumed that Austin hypothesised that a magic multiple (independent of n) can correct (or *mitigate*) the relative error.

This leads to the ***approximate*** **time-interval-update** function with the magic multiple M applied:

$$c_n=M\times approx(n)$$
$$=\begin{cases}M^s\times exact(0)=\frac{M^sf_t}{\bar{\omega}}\left[-\omega_i+\sqrt{\omega_i^2+2\bar{\omega}\alpha}\right], & n=0\\ approx(n-1)-\frac{2\alpha\bar{\omega}\times approx(n-1)}{2\omega_i^2+\alpha\bar{\omega}(4n+1)}, & n>0\end{cases}$$
$$=\begin{cases}M^s\times exact(0)=\frac{M^sf_t}{\bar{\omega}}\left[-\omega_i+\sqrt{\omega_i^2+2\bar{\omega}\alpha}\right], & n=0\\ M^s\times exact(0)\times\prod_{k=1}^{n}\frac{2\omega_i^2+\alpha\bar{\omega}(4k-1)}{2\omega_i^2+\alpha\bar{\omega}(4k+1)}, & n>0\end{cases} \tag{16}$$

As such a new definition for the relative error follows:

$$\text{relative error E}(n)=\left|\frac{M\times approx(n)-exact(n)}{exact(n)}\right| \tag{17}$$

This new definition for the relative error will help determine if including a magic multiple to the ***approximate*** **time-interval-update** function will correct (or *mitigate*) the relative error.

Secondly a test is required to determine when the magic multiple (M) is optimised:

$$\text{Let } F=\sum_{n=0}^{\infty}E^2=\sum_{n=0}^{\infty}\left[\frac{M\times approx(n)-exact(n)}{exact(n)}\right]^2 \tag{18}$$

We want to find M such that F is minimised:

$$\begin{aligned}
0 &= \frac{\partial F}{\partial M} \\
&= 2\sum_{n=0}^{\infty}\left[\frac{M\times approx(n) - exact(n)}{exact(n)}\right]\left[\frac{approx(n)}{exact(n)}\right] \\
&= \sum_{n=0}^{\infty}\left[\frac{M\times approx^2(n) - exact(n)\times approx(n)}{exact^2(n)}\right] \\
&= M\sum_{n=0}^{\infty}\left[\frac{approx(n)}{exact(n)}\right]^2 - \sum_{n=0}^{\infty}\left[\frac{approx(n)}{exact(n)}\right]
\end{aligned} \tag{19}$$

$$\therefore M = \frac{\sum_{n=0}^{\infty}\left[\frac{approx(n)}{exact(n)}\right]}{\sum_{n=0}^{\infty}\left[\frac{approx(n)}{exact(n)}\right]^2} = \frac{1+\sum_{n=1}^{\infty}\left[\frac{approx(n)}{exact(n)}\right]}{1+\sum_{n=1}^{\infty}\left[\frac{approx(n)}{exact(n)}\right]^2} \tag{20}$$

If $approx(n)$ can be converted to a closed function, then Eq. (20) above can be applied quite easily. However it is suspected that the hypothetical closed function would be extremely complicated for a basic low-end microcontroller to implement. Furthermore an infinite summation is not practical.

Consequently an alternative method is needed to determine the magic multiple.

5. Finding the magic multiple using matlab®

A general solution for the magic multiple is no longer attempted in this paper. Instead a general technique is demonstrated to determine a practical solution for the magic multiple. For the remainder of this paper, typical values for a hobby stepper motor are used.

5.1. Typical values used [6]

$$\begin{aligned}
\alpha &= \frac{2\pi}{200}\ \text{rad} = 1.8^\circ \\
f_t &= \frac{16,000,000}{1024} = 15,625 \\
\bar{\omega} &= (0,\quad 10]\quad (\text{in steps of } 0.1) \\
\omega_i &= [0,\quad 7.5]\quad (\text{in steps of } 0.1)
\end{aligned}$$

NB: α and f_t are fixed for any particular stepper motor and microcontroller.

It should be noted that Austin in [1] recommends using the highest frequency possible. A Pre-Scalar of 1024 was used without any noticeable issues. The frequency has no bearing on the relative error however.

It should also be noted that if the particular specifications differ from above, it is then highly recommended to test the analysis independently using a software package.

Since Eq. (20) is either too complex or impractical to implement, it is hypothesised that there exists an approximation of M, denoted M^s, that is more simple or more practical for a basic low-end microcontroller to implement.

I.e. Assume M^s exists such that $\max(\text{abs}(\text{E}(n)|_{M=M^s})) < \gamma$.

The constant γ is simply a subjective arbitrary threshold value used to determine whether the relative error Eq. (17) from using the ***approximate*** **time-interval-update** function with the magic multiple Eq. (16) is deemed satisfactorily '*good*'. It can be assumed that $\gamma = 0.03$ for most intents and purposes.

To find a suitable M^s candidate, it is noted that the highest derivative with respect to n of the $E(n)|_{M=1}$ curve *seems* to occur when $n = 1$ irrespective of ω_i and $\bar{\omega}$. Fig. 2 illustrates this when $\omega_i = 2$ and $\bar{\omega} = 1$.

Since Eq. (15) is a recursive function where future values are based on previous values, it *may be* worthwhile attempting to eliminate the error early on with the hope that the carry-over effect reduces the error throughout. The largest change in relative error occurs at $n = 1$ so it is a good place to begin. From (14)–(16):

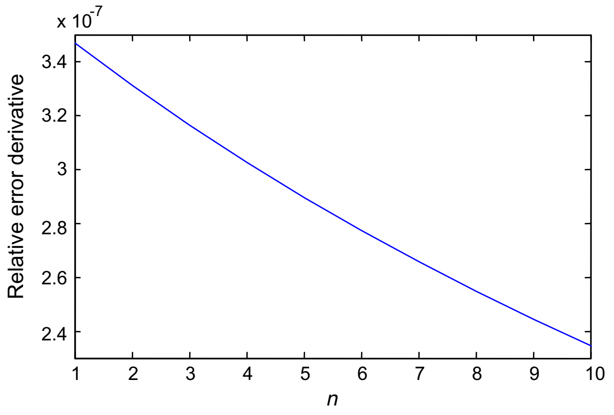

Fig. 2. Derivative of $E(n)|_{M=1}$ with respect to n.

$$exact(1) = \frac{f_t}{\bar{\omega}}\left[\sqrt{\omega_i^2 + 4\bar{\omega}\alpha} - \sqrt{\omega_i^2 + 2\bar{\omega}\alpha}\right]$$

$$approx(1) = \frac{f_t}{\bar{\omega}}\left[\sqrt{\omega_i^2 + 2\bar{\omega}\alpha} - \omega_i\right]\left[\frac{2\omega_i^2 + 3\alpha\bar{\omega}}{2\omega_i^2 + 5\alpha\bar{\omega}}\right]$$

Equating $exact(1) = M^S \times approx(1)$:

$$M^s = \frac{\sqrt{\omega_i^2 + 4\bar{\omega}\alpha} - \sqrt{\omega_i^2 + 2\bar{\omega}\alpha}}{\left[\sqrt{\omega_i^2 + 2\bar{\omega}\alpha} - \omega_i\right]\left[\frac{2\omega_i^2 + 3\alpha\bar{\omega}}{2\omega_i^2 + 5\alpha\bar{\omega}}\right]} \tag{21}$$

To test the accuracy of M^s, it is compared to M using the absolute error $|M - M^s|$. M is calculated by truncating the infinite summation in Eq. (20) to 1001 summations.

It can be seen from Fig. 3 that the maximum absolute error is 0.015947 for the relevant ω_i *and* $\bar{\omega}$ domain. This is acceptably negligible.

It should be noted that the absolute error $|M - M^s|$ varies with ω_i and $\bar{\omega}$ only, since neither M nor M^s are functions of n.

Although M^s may be a close approximate to M, our ultimate aim is for the relative error to be as small as possible. The accuracy of $\text{E}(n)|_{M=M^s}$ can be tested.

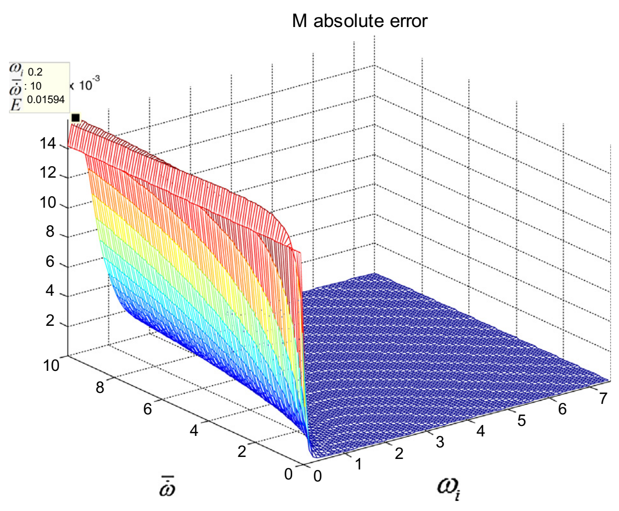

Fig. 3. Absolute error $|M - M^s|$.

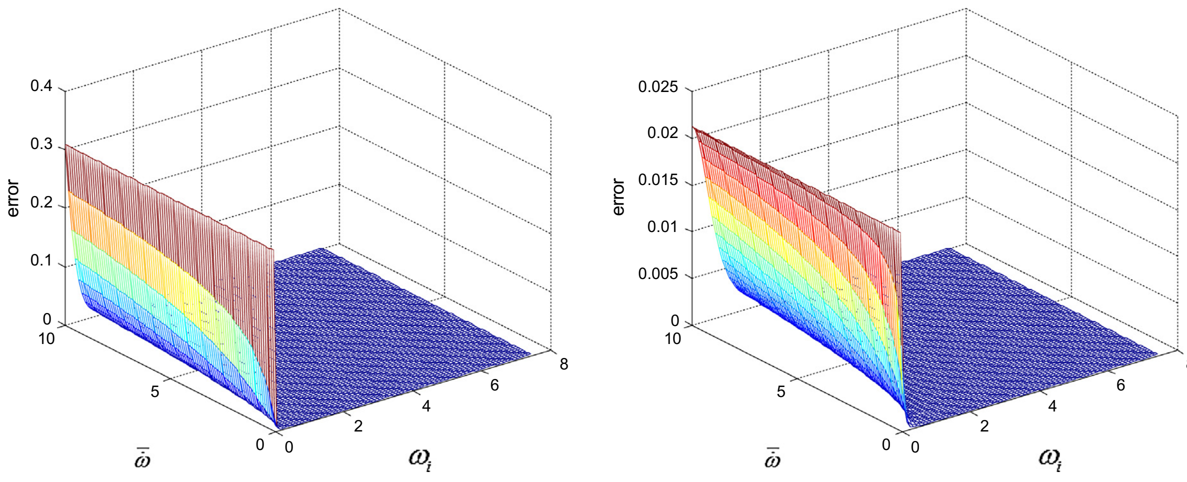

Fig. 4. Relative error $E(n)|_{M=M^s}$ when (a) $n = 0$ and (b) $n = 1001$.

Although M^s may be a close approximate to M, our ultimate aim is for the relative error to be as small as possible. The accuracy of $E(n)|_{M=M^s}$ can be tested.

It can be seen from Fig. 4a that when $n = 0$ the maximum relative error is 0.30964. This appears to be unacceptably large. However, since by definition *approx*(0) = *exact*(0), setting any $M \neq 1$ will distort the relative error.

Since M^s was calculated by equating *exact*(1) = $M^S \times$ *approx*(1), naturally the relative error is 0 when $n = 1$.

The most important result can be seen from Fig. 4b. When $n = 1001$, the maximum relative error seems to converge to 0.021269. It is also the maximum relative error in the range $n = [1, \quad 1001]$. This is below the arbitrary threshold $\gamma = 0.03$.

6. Comparison to results without magic multiple

The previous section shows that incorporating a Magic Multiple (more specifically M^s) reduces the relative error to an acceptable level. However, there is always the possibility that the relative error was already going to be below the threshold $\gamma = 0.03$ even without a Magic Multiple. This can be tested by substituting $M = 1$, which is equivalent to no Magic Multiple being applied.

It can be seen in Fig. 5a that when $n = 1$, the maximum relative error is 0.44853. It can also be seen in Fig. 5b that when $n = 1001$, the maximum relative error *seems* to converge to 0.47934. Both are unacceptably large. In fact throughout the range $n = [1, \quad 1001]$ the maximum relative error is unacceptable.

When compared to the results from the previous section, this demonstrates that accuracy and performance is improved drastically when the ***approximate* time-interval-update** function is changed so that $c_0 = M^s \times$ *approx*(0).

7. Further improvements – initial 'step'

Even when the ***approximate* time-interval-update** function is changed to include the magic multiple, Fig. 4a shows that when $n = 0$, the maximum relative error is unacceptably large. In Section 5, the reason for this was stated but the merits were not discussed.

It must be noted that the Magic Multiple by its derivation in (20) and (21) is *always* a positive fraction less than one.

Austin in [1] states that this is beneficial because it means that the initial step is programmed to 'fire' slightly earlier than it should – from a physics point of view. He asserts that this compensates for the initial inertia of the stepper motor.

Whilst the validity of Austin's claim was not tested, it does *seem* to make sense. However, the results of this paper are generalised so that the stepper motor does not necessarily have to begin from standstill.

Therefore, the first improvement suggested is for $n = 0$ to be treated as a special case. For the first 'step' the Magic Multiple

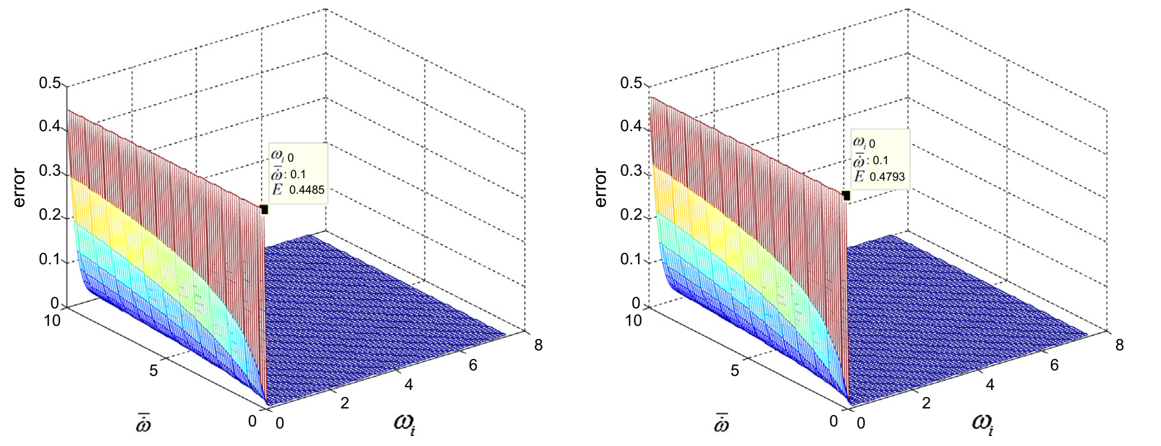

Fig. 5. Relative error $E(n)|_{M=1}$ when (a) $n = 1$ and (b) $n = 1001$.

should not be applied (or alternatively, only applied it if the motor is in standstill). It should only be applied on the second 'step'. This leads to the **new improved *approximate* time-interval-update** function:

$$c_n = improved\_approx(n)$$
$$= \begin{cases} exact(0) = \frac{f_t}{\bar{\omega}}\left[-\omega_i + \sqrt{\omega_i^2 + 2\bar{\omega}\alpha}\right], & n = 0 \\ M^s improved\_approx(n-1) - \dfrac{2\alpha\bar{\omega} \times M^s improved\_approx(n-1)}{2\omega_i^2 + \alpha\bar{\omega}(4n+1)}, & n = 1 \\ improved\_approx(n-1) - \dfrac{2\alpha\bar{\omega} \times improved\_approx(n-1)}{2\omega_i^2 + \alpha\bar{\omega}(4n+1)}, & n > 1 \end{cases}$$
$$= \begin{cases} exact(0) = \frac{f_t}{\bar{\omega}}\left[-\omega_i + \sqrt{\omega_i^2 + 2\bar{\omega}\alpha}\right], & n = 0 \\ M^s exact(0) \times \prod_{k=1}^{n} \dfrac{2\omega_i^2 + \alpha\bar{\omega}(4k-1)}{2\omega_i^2 + \alpha\bar{\omega}(4k+1)}, & n > 0 \end{cases} \tag{22}$$

8. Further improvements – simplification of magic multiple

Currently M^s requires two square root calculations which are computationally expensive. This is similar to the ***exact* time-interval-update** function (14) which justified the derivation of the ***approximate* time-interval-update** function (15).

Unlike (14) which needs to be continually calculated and applied, M^s needs only to be calculated once and applied at the beginning. This is computationally bearable unless the acceleration rate is extremely frequently changed.

If the acceleration rate is required to change frequently, M^s can be further simplified.

Firstly an unacceptability threshold value γ needs to be set. It must realistically suit the intended purpose.

If the relative error exceeds γ, then M^s must be applied. If however γ is not exceeded, then M^s does not need to be applied at all. This can save valuable computation power and time.

The graphs in Fig. 6 show the relative error when M^s is not applied for $n = 2$. The black line represents the intersection

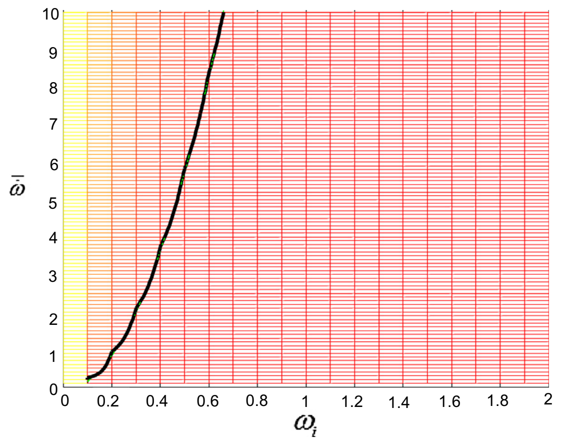

Fig. 7. Relative Error without M^s applied ($\gamma = 0.03$) (birds-eye view: $\omega_i - \bar{\omega}$ plane).

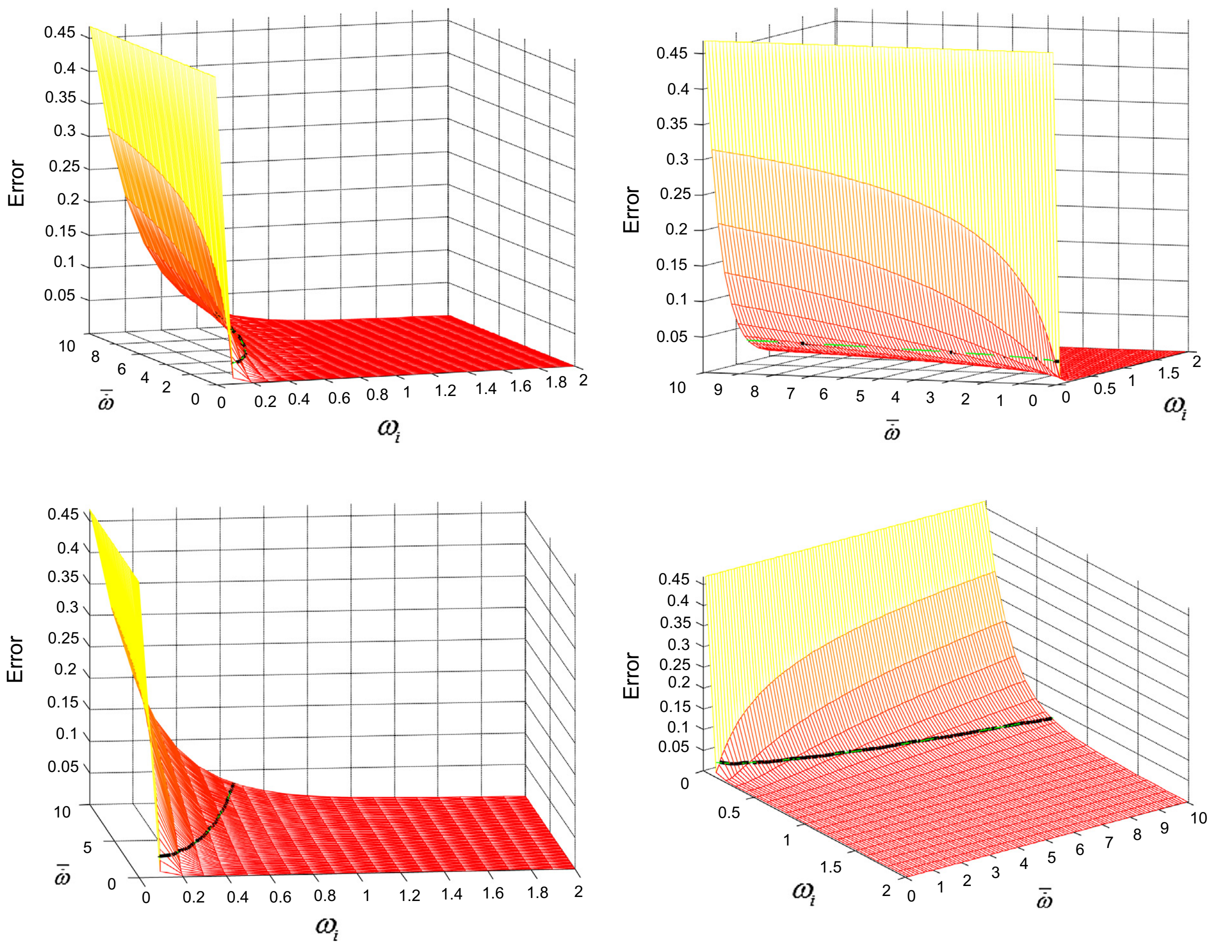

Fig. 6. Relative Error without M^s applied ($\gamma = 0.03$) – various angles.

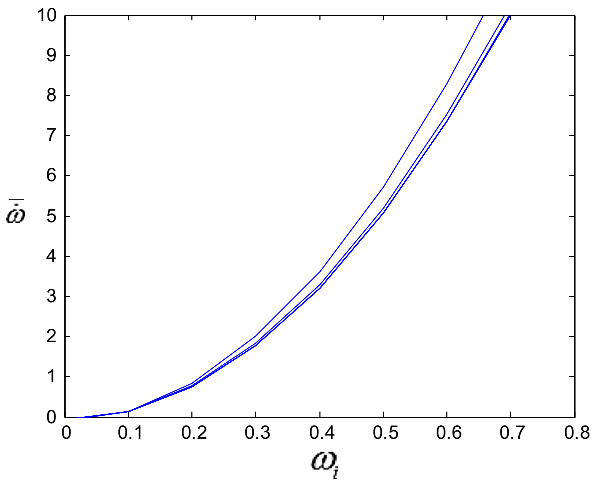

Fig. 8. Plot of the five quadratic lines-of-best-fit.

points between the relative error surface plot and the *relative error* = γ = 0.03 plane. Various angles of the same surface plot are shown.

It was discovered that the black intersection line could be approximated very closely by a quadratic curve.

The green dashed line overlaying the black intersection line is the quadratic line-of-best-fit (see Fig. 7).

Listed below is a sample of quadratic lines-of-best-fit for various n values. Fig. 8 shows the curves in a diagram.

$$\bar{\omega} = 23.3064\omega_i^2 - 0.0539\omega_i - 0.0829, \quad n = 2$$
$$\bar{\omega} = 21.1932\omega_i^2 - 0.0661\omega_i - 0.0702, \quad n = 6$$
$$\bar{\omega} = 20.8097\omega_i^2 - 0.1458\omega_i - 0.0560, \quad n = 50$$
$$\bar{\omega} = 20.8041\omega_i^2 - 0.1473\omega_i - 0.0558, \quad n = 100$$
$$\bar{\omega} = 20.8023\omega_i^2 - 0.1477\omega_i - 0.0557, \quad n = 1000$$

It can be seen from Fig. 8 that as $n \to \infty$ the equation converges. A software package can be used to predetermine the limiting quadratic equation for the relevant ω_i and $\bar{\omega}$ domain.

Once determined, the time-interval-update algorithm can perform a simple logic-test based on the current ω_i and proposed $\bar{\omega}$. If the relative error is unacceptable, then M^s has to be applied. If it is acceptable, then applying M^s can be forgone.

If there is concern that the simple logic-test is too complex, then use of the limiting quadratic equation can be abandoned. Instead, the limiting quadratic equation can provide guidance for an even simpler test such as $\omega_i < 0.7$.

9. Final notes – deceleration

Special care must be taken when $\bar{\omega} < 0$ (deceleration). Before applying Eq. (15), $(\omega_i^2 + 2\bar{\omega}\alpha)$ must be tested for a non-negative value (due to the square root). If it is negative, then the stepper motor direction must be reversed.

If it is initially non-negative, then there is the possibility that c_n could exceed its upper limit (usually indicating acceleration in the reverse direction). This should also be tested and the appropriate action taken. Usually it consists of reversing the stepper motor direction and making the acceleration positive.

10. Conclusion

This paper presented a new algorithm that can be used to generate constant acceleration or deceleration in a stepper motor. A basic low-end microcontroller is more than adequate to implement the algorithm in real time. The algorithm was tested successfully using a small robot programmed to move and change acceleration ‘on the run’. Guidance on implementation of the algorithm is available in [7].

The theoretical foundations for further improving the timing accuracy of the algorithm were also derived. These were not tested using hardware.

It is anticipated that the results of this paper are relevant to various commercial applications such as in toy cars, and devices which require a fine control of stepper motors.

Future improvements are possible – particularly in realizing the new time-interval-update function for FPGA and other hardware based implementations. In fact, Austin’s approach has recently been successfully synthesised for FPGAs in [8–10].

As a final remark, the results of this paper have been patented in Australia [11] with patents possibly pending in other jurisdictions.

References

[1] Austin D. Generate stepper-motor speed profiles in real time, October 9; 2010. <http://www.eetimes.com/design/industrial-control/4006438/Generate-stepper-motor-speed-profiles-in-real-time> EE Times; 30 December 2004. <http://fab.cba.mit.edu/classes/MIT/961.09/projects/i0/Stepper_Motor_Speed_Profile.pdf>.

[2] Jones DW. Control of stepper motors. In: Yeadon WH, Yeadon AW, editors. Handbook of small electric motors. McGraw-Hill; 2001. <http://www.cs.uiowa.edu/ ~ jones/step/> [February 5, 2012].

[3] Mukherjee K, SenGupta S, Bhattacharya TK, Chattopadhyay AK. Development of closed loop control schemes for constant speed operation of a thyristorized commutatorless series motor drive. In: International conference on power electronics and drives systems, 2005 (PEDS 2005), vol. 2; 28–01 November 2005. p. 1112–7.

[4] Hu Qingbo, Lu Zhengyu, Qian Zhaoming. Research on a novel close-loop speed control technique of brushless DC motor. In: Power electronics specialists conference, 2007 (PESC 2007), 17–21 June 2007. IEEE; 2007. p. 2575–8.

[5] Athani VV. Stepper motors: fundamentals* applications and design. New Age International Publishers; 1997.

[6] Typical hobby stepper motor datasheet. <http://www.adafruit.com/datasheets/12vstepper.jpg> [February 5, 2012].

[7] Atmel. AVR446: linear speed control of stepper motor, revision A, 15 p. <http://www.atmel.com/dyn/resources/prod_documents/doc8017.pdf http://www.atmel.com/Images/doc8017.pdf> [February 5, 2012].

[8] Pisi D. Generating stepper motor speed profiles using FPGA. <http://www.feec.vutbr.cz/EEICT/2011/sbornik/03-Doktorske%20projekty/03-Kybernetika%20a%20automatizace/10-xpisid00.pdf> [February 5, 2012].

[9] Tzung-Cheng Chen, Yung-Chun Su. High performance algorithm realization on FPGA for stepper motor controller. In: SICE annual conference, 2008; 20–22 August 2008. p. 1390–5.

[10] Wang Bangji, Liu Qingxiang, Zhou Lei, Zhang Yanrong, Li Xiangqiang, Zhang Jianqiong. Velocity profile algorithm realization on FPGA for stepper motor controller. In: 2nd International conference on artificial intelligence, management science and electronic commerce (AIMSEC), 2011; 8–10 August 2011. p. 6072–5.

[11] Siripala PJ. Generalized solution for generating stepper motor speed profiles in real time. Australian Patent 2011101701, December 30; 2011.